

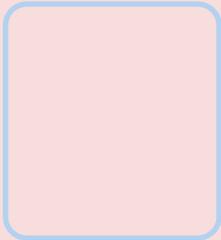
Extension Schemas

Modules Roadmap: You Are Here

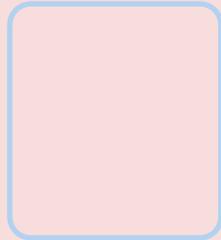
- Anatomy of an XML Exchange
- XML Conceptual Review
- Basic XML Schema for NIEM
- Advanced XML Schema for NIEM
- Substitution Groups
-  Extension Schemas

Objectives Roadmap:

This module supports the following course objective:



Define the physical components of an XML exchange.



Identify basic XML components that are used in the NIEM structure.



Write and/or extend an XML schema conformant to the NIEM Naming and Design Rules (NDR).

Module Objectives

- After completing this module, you should be able to:
 - ◆ Describe real world situations where extension schemas are required.
 - ◆ Explain the three methods of extending a schema.
 - ◆ Extend a complex type from another namespace using the different extension mechanisms.

Definition

- The “X” in XML is for eXtensible.
 - ◆ Exchange Architects can define new tags and attribute names specific to their applications.
 - ◆ If an application is created to extract information from an XML document, the architect of the document can add extra information to it without interfering with the function of the application.

Why would we Extend?

- NIEM is a large data model, but does not describe everything.
 - ◆ NIEM asks that if you use one of their elements, that it be used in exact context.
 - ◆ If the proper definition does not exist in NIEM, create an extension!
- Allows you to design your own customized documents that include NIEM and your own elements in a schema that will validate against the NIEM standard.
- Promotes valuable re-use of extension components from others.

NIEM Schema Extension Basics

**NIEM
Data
Model**

**NIEM
Subset
Schema**

**Exchange
Schema**

Import Subset

Import Extension

**Local
Extension
Schema(s)**

**Extension
Schema**

What are Extensions?

- Locally required Elements (Properties) or Types not precisely represented within NIEM.
- Rare to create an exchange without an extension.
- Extension can add a new “Type” or concept.
- More often used to add elements to an existing Type.

Type Extension

- You can derive one complex type from another.
- The derived type's content model will appear immediately after the base type's model.
- The mechanism to create an extension is very straightforward.
 - ◆ Use "**xs:extension**" schema element to extend a named complex type.

Controlling Type Extensions

- A complex type can be labeled ***abstract*** or ***final***.
- ***Abstract*** means that the type cannot itself be the type of an element – it must be extended first.
- ***Final*** means that the type cannot be extended (by restriction or extension).

```
<xs:complexType name="CT" abstract="true">
```

```
<xs:complexType name="CT" final="restriction">
```

```
<xs:complexType name="CT" final="restriction, extension">
```

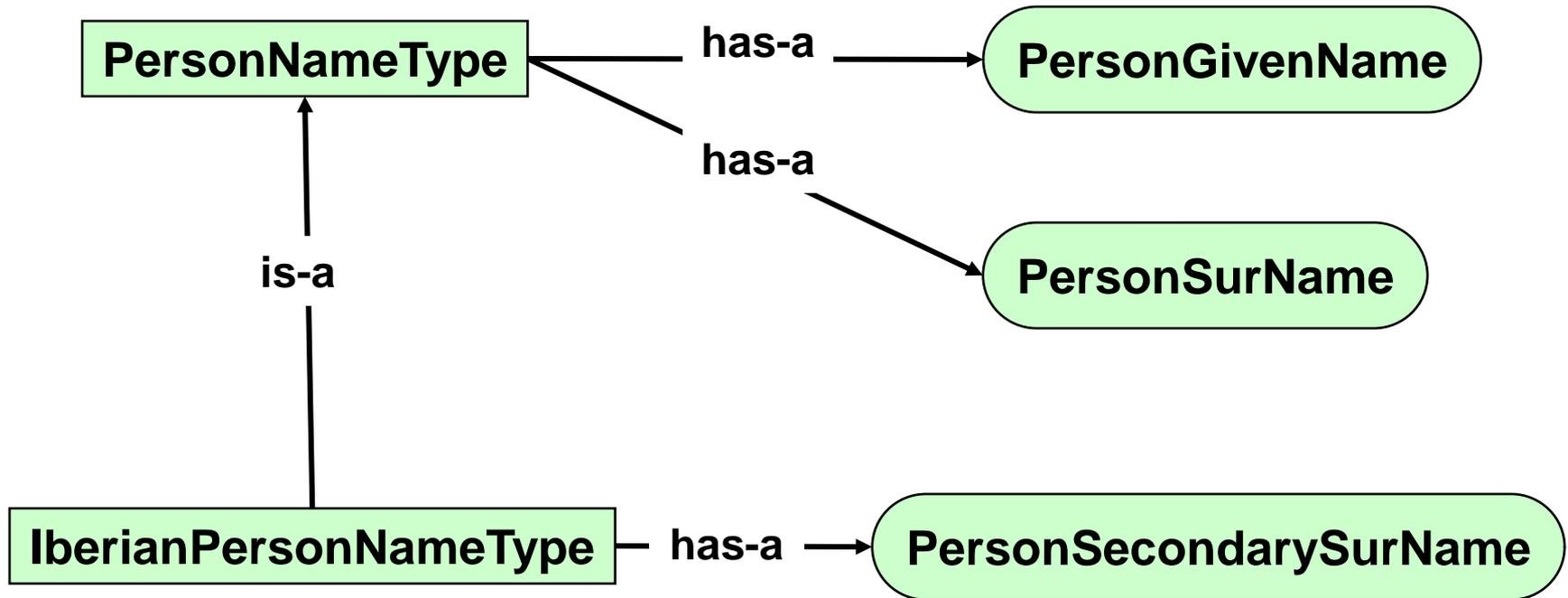
Example Issue – Iberian Surname (1 of 2)

Vicente **Fox** **Quesada**

(Former President of Mexico)

- Individuals from Spanish speaking countries often have two (2) surnames:
 - ◆ (1) from Father's side (1) from Mother's side.
- **Fox** is not a middle name, but his primary surname, from his father. (“President Fox”)
- **Quesada** is his 2nd surname, from his mother.
- NIEM supports only (1) PersonSurname.

Example Issue – Iberian Surname (2 of 2)



Creating the Type Extension

```
<xs:complexType name="IberianPersonNameType">
  <xs:complexContent>
    <xs:extension base="nc:PersonNameType">
      <xs:sequence>
        <xs:element name="PersonSecondarySurName"
          type="nc:PersonNameTextType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Create the new type

Use the xs:extension element and specify which type is being extended

Add the pieces that the extension needs

Using Type Extensions

- Type extension can be implemented three (3) ways:
 - ◆ Implied Substitution
 - ◆ Concrete Substitution
 - ◆ Dynamic Substitution
 - Used in GJXDM, but seldom used in NIEM

Implied Substitution Group

- Involves making a new type in XML Schema, based on existing NIEM types.
 - ◆ New element is tied to the element to be replaced using the **substitutionGroup** attribute.
- For example, using the "IberianPersonNameType" from previous slide,
 - ◆ Create element called "PersonNameType"
 - ◆ Adding the "substitutionGroup" attribute referencing "nc:PersonName"

Implied Substitution Group Schema

```
<xs:complexType name="IberianPersonNameType">  
  <xs:complexContent>  
    <xs:extension base="nc:PersonNameType">  
      <xs:sequence>  
        <xs:element name="PersonSecondarySurName"  
          type="nc:PersonNameTextType" />  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>
```

The type from
the previous
slide

Create new
element

```
<xs:element name="PersonName"  
  type="local-ns:IberianPersonNameType"  
  substitutionGroup="nc:PersonName" />
```

Define element as
allowable
substitution for
the extended type

Implied Substitution Group Instance Document

Substitution Group attribute explicitly allows extended PersonName to be used instead of the parent

<nc:Person>

<local-ns:PersonName>

<nc:PersonGivenName>Gabriel</nc:PersonGivenName>

<nc:PersonSurName>Garcia</nc:PersonSurName>

<local-ns:PersonSecondarySurName>

Marquez

</local-ns:PersonSecondarySurName>

</local-ns:PersonName>

</nc:Person>

Substitution Groups—Caveat

- Extending NIEM substitution groups should only occur if the newly created element is semantically equivalent to the substitution group's head element.
- Don't just do it because its a simpler form of extension.

Concrete Extension

- Concrete Extension involves creating a new type by extending existing NIEM type in XML Schema to include additional local elements.
 - ◆ Then, in an instance document, elements of this new type are now in a local namespace, along with higher level elements that contain them.
- For example, using the "IberianPersonNameType" from earlier:
 - ◆ In your instance document, "PersonSecondarySurName" would be in your local namespace.
 - ◆ Additionally, the "PersonName" and "Person" will also be in your local namespace.

Concrete Extension—Schema

```
<xs:complexType name="IberianPersonNameType">
  <xs:complexContent>
    <xs:extension base="nc:PersonNameType">
      <xs:sequence>
        <xs:element name="PersonSecondarySurName" type="nc:PersonNameTextType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="IberianPersonType">
  <xs:complexContent>
    <xs:extension base="nc:PersonType">
      <xs:sequence>
        <xs:element name="PersonIberianName"
          type="local-ns:IberianPersonNameType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Need to extend person
as well so the explicit
type can be referenced
in the XML instance

Concrete Extension—Instance

```
<local-ns:IberianPerson>  
  <local-ns:PersonIberianName>  
    <nc:PersonGivenName>Gabriel</nc:PersonGivenName>  
    <nc:PersonSurName>Garcia</nc:PersonSurName>  
    <local-ns:PersonSecondarySurName>  
      Marquez  
    </local-ns:PersonSecondarySurName>  
  </local-ns:PersonIberianName>  
</local-ns:IberianPerson>
```

Instance explicitly references the extended type

Dynamic Type Substitution

- Dynamic Type Substitution involves creating a new type in XML Schema, based on existing NIEM types.
 - ◆ Then, in an instance document, an element is reassigned to that new type via the **xsi:type** construct to substitute the new type for the usual type
- For example, using the "IberianPersonNameType" from earlier:
 - ◆ In the instance document, you would use a "PersonName" object but declare its type to be "IberianPersonNameType" instead.

Dynamic Type Substitution—Schema

This type gets specified in the instance document

```
<xs:complexType name="IberianPersonNameType">
  <xs:complexContent>
    <xs:extension base="nc:PersonNameType">
      <xs:sequence>
        <xs:element name="PersonSecondarySurName"
          type="nc:PersonNameTextType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Dynamic Type Substitution—Instance

Indicates the type that
PersonName really is

```
<nc:PersonName xsi:type=  
    "local-ns:IberianPersonNameType">  
  <nc:PersonGivenName>Gabriel</nc:PersonGivenName>  
  <nc:PersonSurName>Garcia</nc:PersonSurName>  
  <local-ns:PersonSecondarySurName>  
    Marquez  
  </local-ns:PersonSecondarySurName>  
</nc:PersonName>
```

Can use this element
because xsi:type was
specified above

Extension Method Comparison

	Implied Substitution	Concrete	Dynamic
Web Services	Yes	Yes	No
Defined in schema	Yes	Yes	No
Readability	Moderate	Moderate	Hardest
Implementation Level of Effort	Easy	Difficult	Easy
Ease of Validation	Easy	Easy	Difficult
Elements in local namespace	Fewest	Most	Fewest
Semantic Precision	Most	Least	Most

Exercise 6.1: Implied Substitution

- Using the substitution schema on the next slide, create an instance document with the following values:
 - ◆ VehicleID = “WFM30988EFC2981”
 - ◆ VehicleMake = “Ford”
 - ◆ VehicleModel = “Escape GL”
 - ◆ VehicleFuelCellVoltage = 48.8

Exercise 6.1: Implied Substitution

```
<xs:complexType name="HybridVehicleType">
  <xs:complexContent>
    <xs:extension base="nc:VehicleType">
      <xs:sequence>
        <xs:element name="VehicleFuelCellVoltage"
          type="nc:NumericType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="Vehicle"
  type="local-ns:HybridVehicleType"
  substitutionGroup="nc:Vehicle" />
```

Solution 6.1: Implied Substitution

```
<local-ns:Vehicle>  
  <nc:VehicleID> WFM30EFC2981 </nc:VehicleID>  
  <nc:VehicleMake>Ford</nc:VehicleMake>  
  <nc:VehicleModel>Escape GL</nc:VehicleModel>  
  <local-ns:VehicleFuelCellVoltage>  
    48.8  
  </local-ns:VehicleFuelCellVoltage>  
</local-ns:Vehicle>
```

Exercise 6.2: Concrete Substitution

- Using the concrete substitution schema on the next slide, create an instance document with the following values:
 - ◆ PersonGivenName = **Marge**
 - ◆ PersonSurname = **Simpson**
 - ◆ PersonSecondarySurname = **Bouvier**

Exercise 6.2: Concrete Type Extension

```
<complexType name="PersonNameType">  
  <complexContent>  
    <extension base="nc:PersonNameType">  
      <sequence>  
        <element name="PersonSecondarySurName" type="nc:PersonNameTextType" />  
      </sequence>  
    </extension>  
  </complexContent>  
</complexType>
```

```
<element name="Person">  
  <complexType>  
    <complexContent>  
      <extension base="nc:PersonType">  
        <sequence>  
          <element name="PersonName" type="local-ns:PersonNameType" />  
        </sequence>  
      </extension>  
    </complexContent>  
  </complexType>
```

Solution 6.2: Concrete Substitution

<local-ns:Person>

<local-ns:PersonName>

<nc:PersonGivenName>Marge</nc:PersonGivenName>

<nc:PersonSurName>Simpson</nc:PersonSurName>

<local-ns:PersonSecondarySurName>

Bouvier

</local-ns:PersonSecondarySurName>

</local-ns:PersonName>

</local-ns:Person>

Final Thoughts on Extensions

- Most often, the mechanism for use will be the simplest to implement:
 - ◆ Simple extensions for adding local code values.
 - ◆ Implied substitution favored over concrete extensions.
 - ◆ Extensions are not something to be avoided.
 - ◆ Extensions are foundational to NIEM and acknowledge the need for local flexibility.

Module Summary

- After completing this module, you should be able to:
 - ◆ Describe real world situations where extension schemas are required.
 - ◆ Explain the three methods of extending a schema.
 - ◆ Extend a complex type from another namespace using the different extension mechanisms.

Objectives Summary:

By the end of Day 1, you will be able to:



Define the physical components of an XML exchange.



Identify basic XML components that are used in the NIEM structure.



Write and/or extend an XML schema conformant to the NIEM Naming and Design Rules (NDR).

Creative Commons



Attribution-ShareAlike 2.0

You are free to

- Copy, distribute, display, and perform the work
- Make derivative works
- Make commercial use of the work

Under the following conditions

- For any reuse or distribution, you must make clear to others the license terms of this work
- Any of these conditions can be waived, if you get permission from the copyright holder

Your fair use and other rights are in no way affected by the above

This is a human-readable summary of the [Legal Code \(the full license\)](#) and [Disclaimer](#)

This page is available in the following languages

[Català](#), [Deutsch](#), [English](#), [Castellano](#), [Suomeksi](#), [français](#), [hrvatski](#), [Italiano](#), [日本語](#), [Nederlands](#), [Português](#), and [中文\(繁\)](#)

[Learn how to distribute your work using this license](#)



Attribution—You must give the original author credit



ShareAlike—If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one